Communications Research Centre Canada
Centre de recherches sur les communications Canada

A Reliable Transport Protocol for Resource Constrained Nodes: CRCTP - Protocol Design

Un Protocole de transport avec garantie de livraison pour les appareils de communications aux ressources limitées : CRCTP – Conception du protocole

Prepared by :
Mathieu Déziel, Eng.
Avril 2014

CRC Technical Note No. CRC-TN-2014-001
Note Technique du CRC # CRC-TN-2014-001

Prepared for:
David Waller
Defence Scientist
Telephone: (613) 998-9985

The scientific or technical validity of this Contract Report is entirely the responsibility of the Contractor and the contents do not necessarily have the approval or endorsement of the Department of National Defence of Canada.

ATTENTION Ces renseignements sont fournis à la condition expresse que les droits de propriété et les droits de brevet soient protégés.	CAUTION This information is provided with the express understanding that proprietary and patent rights will be protected

DRDC-RDDC-2014-C109
June 2014

Canada

Abstract

During the scope of two DRDC projects related to wireless sensor networks (SASNet, Self-Healing Autonomous Network, 2007-2011 and WRSN, Wireless Radiation Sensor Network, 2012-2014), CRC designed a transport protocol called CRCTP (CRC Transport Protocol).
A Transport Protocol for Resource Constrained Nodes: CRCTP - Protocol Design
The two de-facto transport protocols typically found in IP networks are TCP (Transmission Control Protocol) and UDP (User Datagram Protocol). UDP does not provide guarantee of delivery, but it is useful when low latency is required. TCP does provide guaranteed delivery. The mechanism that TCP uses to provide its guaranty of delivery operates end-to-end, i.e. only the two communicating endpoints are involved. This means that a failed transmission, even though it fails at the last hop before the destination, is restarted completely. Also, TCP uses a congestion control mechanism that causes unnecessary delays in wireless networks, and particularly in multi-hop networks. These two reasons make TCP inefficient under some circumstances, and the latency introduced is unacceptable for some applications.

CRCTP was designed with the objective of having a transport protocol that provides guaranteed delivery with the lowest latency and overhead possible. It is designed to operate on resource constrained nodes and networks with restricted bandwidth. Various techniques are used to lower the overhead, latency and memory consumption, such as: hop-by-hop delivery guarantee mechanism, hybrid ACK and NACK confirmation, no congestion control at all, etc.

This document presents the design of CRCTP. The design is generic such that the protocol can be implemented in most typical communications stack.

Résumé

Dans le cadre de deux projets de RDDC (SASNet, Self-Healing Autonomous Network, 2007-2011 et WRSN, Wireless Radiation Sensor Network, 2012-2014) étudiants des réseaux de capteurs sans fils, le CRC a fait la conception d'un protocole de transport appelé CRCTP (CRC Transport Protocol).

On retrouve deux protocoles de transport qui sont largement utilisés dans les réseaux IP communs : TCP (Transmission Control Protocol) et UDP (User Datagram Protocol). UDP est un protocole qui ne fournit pas de garantie de livraison, mais il est fort utile quand l'application demande des transmissions de données avec de courts délais. À l'inverse, TCP fournit une garantie de livraison. Le mécanisme qu'utilise TCP pour confirmer la livraison des paquets fonction bout-à-bout, c'est-à-dire que seules la source et la destination sont impliquées. Donc, si un paquet n'est pas reçu correctement lors de la dernière retransmission avant la destination, tout est à recommencer, de la source à la destination finale. De plus, TCP utilise un mécanisme de contrôle de la congestion du réseau qui cause des délais inutiles sur les réseaux sans fil, et particulièrement dans les réseaux multi-sauts. Ces deux raisons font que TCP est particulièrement inefficace sous certaines circonstances, et les délais qui en résultent ne sont pas acceptables pour certaines applications.

CRCTP a été conçu avec comme objectif premier d'avoir un protocole de transport qui fournit une garantie de livraison tout en gardant les délais courts, et en minimisant les coûts sur le réseau. Il a été conçu pour opérer sur des appareils qui sont limités en ressources, et sur des réseaux limités en bande passante. Plusieurs techniques ont été utilisées pour minimiser les coûts sur le réseau, le délai et la demande de mémoire, tel que : mécanisme de confirmation qui opère entre chaque sauts (plutôt que bout-à-bout), mécanisme hybride de ACK et NACK, pas de contrôle de la congestion, etc.

Ce document présente la conception du protocole CRCTP. La conception est générique de sorte que le protocole pourrait être implanté dans la plupart des systèmes de communication communs.

Executive Summary

Background: During the scope of two DRDC projects related to wireless sensor networks (SASNet, Self-Healing Autonomous Network, 2007-2011 and WRSN, Wireless Radiation Sensor Network, 2012-2014), CRC designed a transport protocol called CRCTP (CRC Transport Protocol).

The two de-facto transport protocols typically found in IP networks are TCP (Transmission Control Protocol) and UDP (User Datagram Protocol). UDP does not provide guarantee of delivery, but it is useful when low latency is required. TCP does provide guaranteed delivery. The mechanism that TCP uses to provide its guaranty of delivery operates end-to-end, i.e. only the two communicating endpoints are involved. This means that a failed transmission, even though it fails at the last hop before the destination, is restarted completely. Also, TCP uses a congestion control mechanism that causes unnecessary delays in wireless networks, and particularly in multi-hop networks. These two reasons make TCP inefficient under some circumstances, and the latency introduced is unacceptable for some applications.

CRCTP was designed with the objective of having a transport protocol that provides guaranteed delivery with the lowest latency and overhead possible. It is designed to operate on resource constrained nodes and networks with restricted bandwidth. Various techniques are used to lower the overhead, latency and memory consumption, such as: hop-by-hop delivery guarantee mechanism, hybrid ACK and NACK confirmation, no congestion control at all, etc.

This document presents the design of CRCTP. The design is generic such that the protocol can be implemented in most typical communications stack

Results: This is a protocol design document, as such there is not result presented.

Significance: This document presents the design of a transport protocol. It was designed in the scope of two DRDC projects related to wireless sensor networks (SASNet, Self-Healing Autonomous Network, 2007-2011 and WRSN, Wireless Radiation Sensor Network, 2012-2014). During the WRSN project, an implementation of CRCTP in the Qualnet network simulator was done. CRCTP has the potential to provide guaranteed delivery at low overhead and latency cost – these characteristics make the protocol attractive for a variety of networks, in particular wireless sensor networks.

Future Plans: Although no follow-on project is presently planned, the logical next step is to refine the protocol design based on hands-on experience in simulation, and to port the CRCTP simulation implementation to an implementation on a real device.

Sommaire

Introduction ou contexte : Dans le cadre de deux projets de RDDC (SASNet, Self-Healing Autonomous Network, 2007-2011 et WRSN, Wireless Radiation Sensor Network, 2012-2014) étudiants des réseaux de capteurs sans fils, le CRC a fait la conception d’un protocole de transport appelé CRCTP (CRC Transport Protocol).

On retrouve deux protocoles de transport qui sont largement utilisés dans les réseaux IP communs : TCP (Transmission Control Protocol) et UDP (User Datagram Protocol). UDP est un protocole qui ne fournit pas de garantie de livraison, mais il est fort utile quand l’application demande des transmissions de données avec de courts délais. À l’inverse, TCP fournit une garantie de livraison. Le mécanisme qu’utilise TCP pour confirmer la livraison des paquets fonction bout-à-bout, c'est-à-dire que seules la source et la destination sont impliquées. Donc, si un paquet n’est pas reçu correctement lors de la dernière retransmission avant la destination, tout est à recommencer, de la source à la destination finale. De plus, TCP utilise un mécanisme de contrôle de la congestion du réseau qui cause des délais inutiles sur les réseaux sans fil, et particulièrement dans les réseaux multi-sauts. Ces deux raisons font que TCP est particulièrement inefficace sous certaines circonstances, et les délais qui en résultent ne sont pas acceptables pour certaines applications.

CRCTP a été conçu avec comme objectif premier d’avoir un protocole de transport qui fournit une garantie de livraison tout en gardant les délais courts, et en minimisant les coûts sur le réseau. Il a été conçu pour opérer sur des appareils qui sont limités en ressources, et sur des réseaux limités en bande passante. Plusieurs techniques ont été utilisées pour minimiser les coûts sur le réseau, le délai et la demande de mémoire, tel que : mécanisme de confirmation qui opère entre chaque sauts (plutôt que bout-à-bout), mécanisme hybride de ACK et NACK, pas de contrôle de la congestion, etc.

Ce document présente la conception du protocole CRCTP. La conception est générique de sorte que le protocole pourrait être implanté dans la plupart des systèmes de communication communs.

Résultats : Ce document est un document de conception d’un protocole de transport, donc aucun résultat n’est fourni dans le cadre de ce document.

Importance : Ce document présente la conception d’un protocole de transport. Il a été conçu dans le cadre de deux projets de DRDC reliés aux réseaux de capteurs sans fils (SASNet, Self-Healing Autonomous Network, 2007-2011 et WRSN, Wireless Radiation Sensor Network, 2012-2014). Durant le projet WRSN, une implantation de CRCTP a été faite dans le simulateur de réseau Qualnet. CRCTP a le potentiel d’offrir une garantie de livraison à faible coût pour le réseau, tout en gardant les délais courts. Ces caractéristiques le rendent attrayant pour une variété de réseaux, notamment les réseaux de capteurs sans fils.

Perspectives : Bien qu’il n’y ait pas de suite immédiate à ce projet, la suite logique serait de raffiner la conception du protocole en se basant sur l’expérience pratique obtenue lors des simulations, eu aussi de transférer l'implantation du simulateur vers un vrai appareil de communication.

TABLE OF CONTENT

TABLE OF FIGURES

Acknowledgements

The work reported herein was supported by Defence Research and Development Canada (DRDC).

1 Introduction

In recent years, sensor networks have been widely studied in the research community. Miniaturization of processing units combined with increases in processing speed, improvements in battery technologies and advances in low power wireless technologies have enabled the feasibility to develop innovative networks composed of tens or hundreds of small sensors forming an ad hoc network.

Such networks have constraints that are typically not found in regular wired or wireless networks: low memory devices, low processing speed devices, energy-constrained devices, etc. In the IETF jargon, there is a name for such networks: L2N, for “low power, lossy networks”. In this context, the term “low power” is used to describe the devices of the network, and “lossy” is used to qualify the links of the network. All these characteristics of L2N force designers to re-think the networking aspect of communications (addressing, routing protocol, MAC and link layer protocol, etc).

The two de-facto transport protocols typically found in IP networks are TCP [3] (Transmission Control Protocol) and UDP [2] (User Datagram Protocol). UDP does not provide guarantee of delivery, but it is useful when low latency is required. TCP does provide guaranteed delivery. The mechanism that TCP uses to provide its guaranty of delivery operates end-to-end, i.e. only the two communicating nodes are involved. This means that a failed transmission, even though it fails at the last hop before the destination, it restarted completely. Also, TCP uses a congestion control mechanism that causes unnecessary delays in wireless networks, and particularly in multi-hop networks. These two reasons make TCP inefficient under some circumstances, and the latency introduced is unacceptable for some applications.

Earlier, we surveyed transport protocol for sensor networks [1]. We investigated more thoroughly one protocol that had met many requirements of wireless sensor networks: PSFQ. PSFQ, although a good protocol for some sensor network applications, consumed a lot of memory. In this document, based on our previous work and experience, we design a new transport protocol suitable for sensors nodes with constrained resources. It provides reliability for both unicast and broadcast/multicast traffic.

The new protocol was designed with the objective of having a transport protocol that provides guaranteed delivery with the lowest latency and overhead possible. It is designed to operate on resource constrained nodes and networks with restricted bandwidth. Various techniques are used to lower the overhead, latency and memory consumption, such as: hop-by-hop delivery guarantee mechanism, hybrid ACK and NACK confirmation, no congestion control at all, etc.

This document starts directly with the design of the new protocol. The purpose of this document is to describe in details the protocol. Care is taken to keep the protocol generic so it can be implemented in the network protocol stack of any network. The design of this protocol was initiated in a DRDC project called SASNet (Self-Healing Autonomous Sensor Network, 2007-2011), and was completed in a follow-on smaller project called WRSN (Wireless radiation Sensor Network, 2012-2014)

For the remainder of this document, we will call the designed transport protocol CRCTP (CRC Transport Protocol).

2 CRCTP Protocol Design

This section will present the complete design of CRCTP.

2.1 Overview

The main purpose of a transport protocol is to provide a transparent transfer of packets between the originator and the destination. It may or may not provide a guarantee that packets are received correctly at the other end – making it a reliable or unreliable transport protocol.

Additional roles of a transport protocol may include fragmentation and reassembly, flow control, port service and congestion control. In the ideal OSI layered network model, the transport protocol is located between session and network layer. This is illustrated in Figure 1.

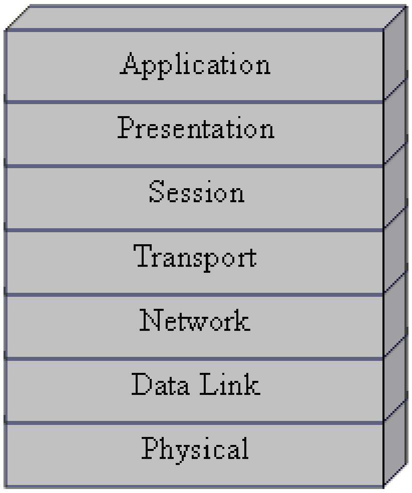


Figure 1 - OSI Layered Network Stack

CRCTP provides the following capabilities:

- Reliable Transport
- Unreliable Transport
- Fragmentation/Reassembly of Packets too large to fit in a single MAC Frame
- Ports Services

Providing reliability is the key challenge; the other capabilities are simpler to implement. Reliability is normally easy to provide. However reliability is a real challenge in our case because of very low bandwidth links with potentially high error rates, and also because the nodes are CPU, memory and power constrained.

Fragmentation/Reassembly introduces some overhead. With CRCTP, this overhead is simultaneously used to provide reliability. This improves overall overhead.

CRCTP provides no congestion control – this is left to other layers of the networking stack. Practically, if CRCTP cannot receive a new packet from a higher layer, the higher layer is made aware of it. This is a simple and crude form of congestion control.

There are different ways of classifying reliable transport protocols. One of them is whether they provide end-to-end or hop-by-hop reliability. TCP is a very popular end-to-end reliable transport protocol. End-to-end protocols perform poorly on multi-hop networks that have links where frame losses are likely to occur. The reason is that if a frame is lost on just one hop, the transmission of that frame is restarted from the originator (the source) of the frame, causing a waste of network resources. Moreover, the higher the error rate is the worse the situation becomes. Hop by hop is therefore a better solution for these networks. Therefore, CRCTP is a hop-by-hop reliable transport protocol.

The danger with hop-by-hop protocol is that when an originator node sends a frame and it is convinced that the next hop received it, it concludes that the whole transmission up to the final destination is successful. But what if something unexpected happens later as the frame travels toward its final destination (lost route, a relay node becoming isolated, etc)? The responsibility for relay nodes is high. CRCTP must be robust enough to handle this situation.

Another method of classifying reliable transport protocols is whether they use positive acknowledgement (typically called ACK) or negative acknowledgement (NACK). A positive acknowledgement is requested by a sender, to confirm the reception of a frame previously sent. A negative acknowledgement is sent by a receiver that detected a transmission error. The advantage of negative acknowledgement is that only erroneous situations cause overhead. With positive acknowledgement, every successful frame cause overhead (ACK) and erroneous situations are detected on failure of reception of an ACK. This is more costly in term of overhead but errors are immediately detected. The disadvantage of NACK is its inability to cope with small (i.e. packet size = 1 frame payload or less) packets, or to detect errors in the last fragment of a multiple-fragments packet. Also, with NACK, it takes a successful transmission to detect previous errors: this may lead to long frame error recovery time in some cases.

CRCTP provides reliability through a hop-by-hop mechanism that uses a hybrid combination of ACK and NACK. It achieves its reliability by doing in-sequence forwarding, and it supports unicast, broadcast and multicast.

2.2 Position in the Network Architecture

As seen in Figure 1, ideally the transport protocol is between the session and network layers. This is not always feasible in practice. For example, with TCP, traditionally TCP works dependently with IP (a network protocol), and the result is called the TCP/IP protocol suite, where IP addresses are used by the transport protocol to identify flows.

Like TCP, CRCTP also needs to be addresses aware (source, final destination, previous hop and next hop addresses). In addition, CRCTP needs to examine each incoming data frame from the lower layers, even if these frames do not finish their network journey in the local node. This is because CRCTP provides hop-by-hop reliability. Therefore, missed frames need to be identified at every hop when a packet travels, so appropriate action(s) are taken. This is why CRCTP, although strictly speaking a transport protocol, is not totally independent from the network layer.

Figure 2 shows how the OSI stack is modified to accommodate CRCTP. Figure 3 shows how CRCTP is implemented in the typical networking stack.

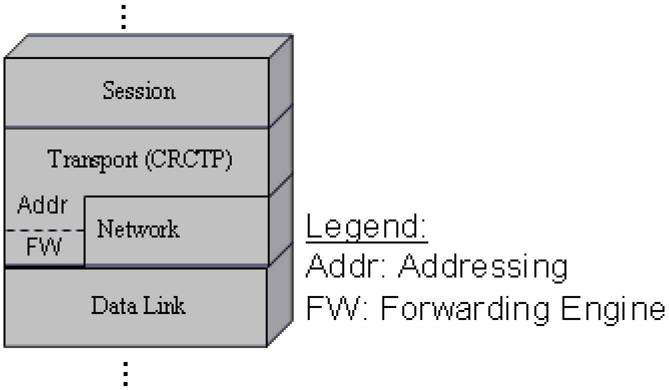


Figure 2 - OSI Modified to Accommodate CRCTP

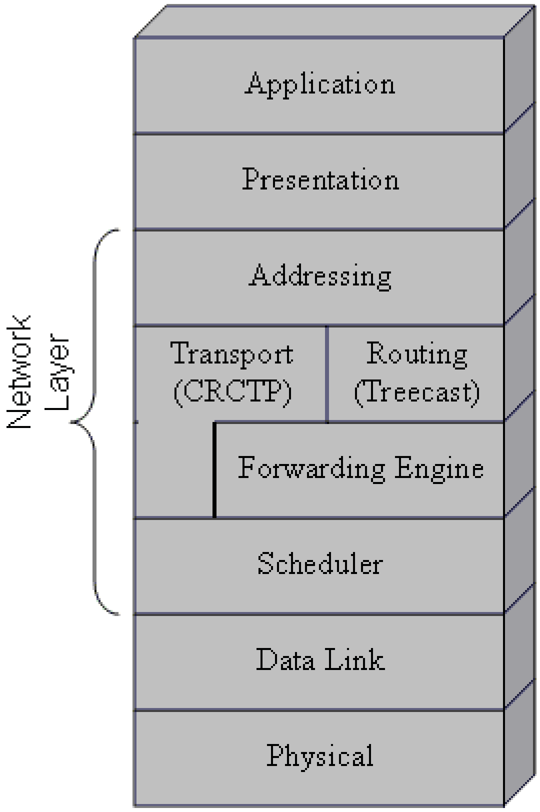


Figure 3 - CRCTP in a typical Networking Stack

2.3 Data Frame Format

The following terminology is used throughout this document. A packet arrives from the higher layers and it enters the transport protocol. Once inside the transport protocol, it is called a transport layer frame, or simply a frame. A large packet is fragmented into multiple frames by the transport protocol: those frames can also be called fragments. Transport layer frames are constructed to fit into a single MAC layer frame. Therefore, at both layers (transport and data link), it's called a frame (transport layer frames are the same as MAC frames, but the MAC layer adds its own header and footer to them).

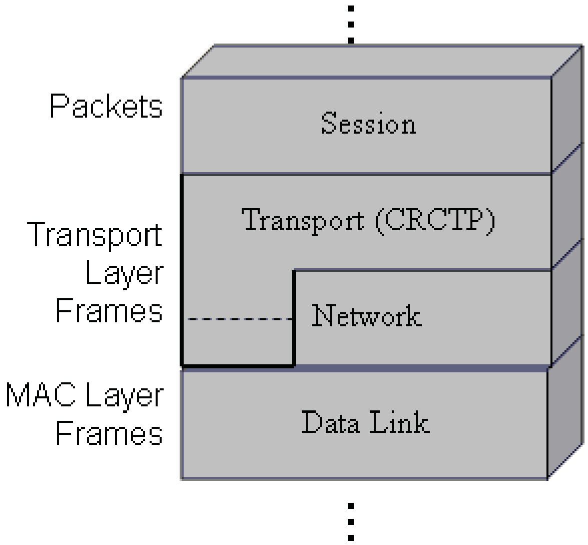
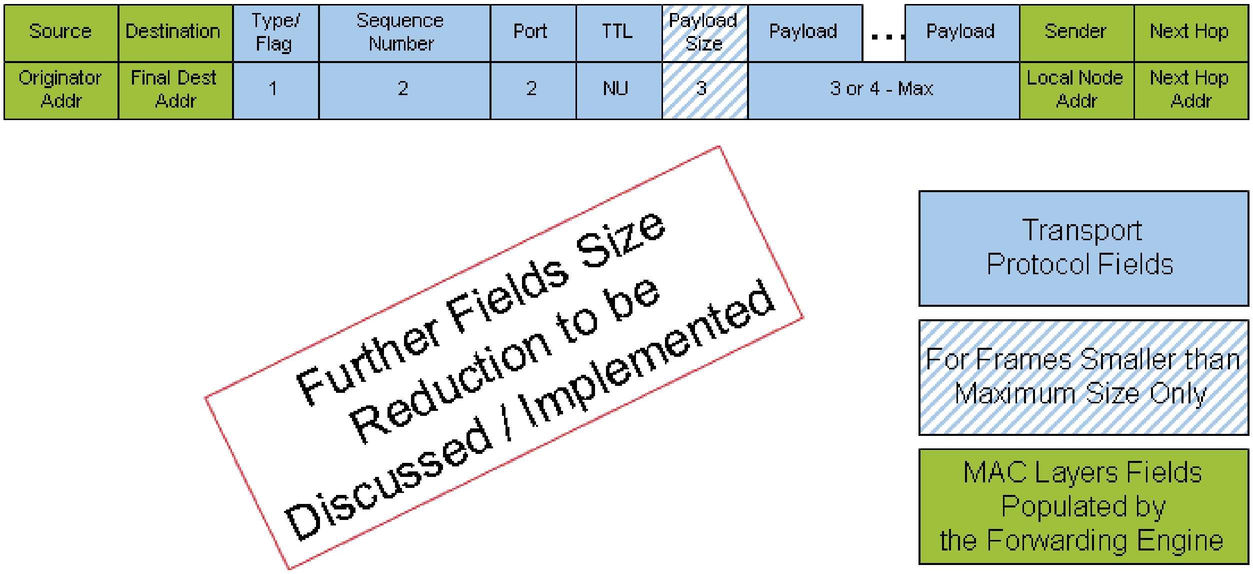


Figure 4 - From Packets to Frames

The data frame format of CRCTP is illustrated in Figure 5. The fields shown in green are actually addressing and MAC layers fields, but they are used by CRCTP to identify flows and missing frames. The next hop address is a field that is populated by the forwarding engine, with the information from the routing protocol. Table 1 explains each field of CRCTP frame.



Source	Destination	Type/ Flag	Sequence Number	Port	TTL	Payload Size	Payload	...	Payload	Sender	Next Hop
Originator Addr	Final Dest Addr	1	2	2	NU	3	3 or 4 - Max			Local Node Addr	Next Hop Addr

Further Fields Size Reduction to be Discussed / Implemented
Transport Protocol Fields
For Frames Smaller than Maximum Size Only
MAC Layers Fields Populated by the Forwarding Engine

Figure 5 - CRCTP General Frame Format

As seen in Figure 5, CRCTP requires information from other layers of the communication stack to operate. This requires what is called a cross-layer design. Cross-layer design is a technique often used in wireless networks where the layers of the communication stack, normally independent from each other, share information to optimize the performance of the protocols. This document does not provide information on how to implement such cross-layer design because it is dependent on specific stack. The exact "how to" would typically be done at the software design stage.

Field	**Size**	**Bits**	**Function**
Source	TBD		The address of the source node, i.e. the node that generated the frame. The value of this field remains unchanged as the frame travels in the network.
Destination	TBD		The final destination address, i.e. the address of the node(s) that are intended receiver(s). The value of this field remains unchanged as the frame travels in the network. Multicast and broadcast frames are identified by the content of this field.
Type/ Mode/ Flags	1 Byte	Type (B1)	
		0	Data Frame
		1	Control Frame
		Mode (B2) (for data frames only)	
		0	Unreliable

Field	Size	Bits	Function
		1	Reliable
		Flags (B3B4B5B6) (for all data frames)	
		B3	Set: First Fragment
		B4	Set: Last Fragment
		B5	Set: Full Size Frame
		B6	Set: New Topology Indication
		Flags (B7B8) (for reliable data frames only)	
		B7	Set: More Frames
		B8	Set: ACK REQ
		Flags (B7B8) (for unreliable data frames only)	
		B7	Set: A Fragment
		B8	Unused
		Control Frame Sub-Type (B2B3B4) (Control Frames Only)	
		000	ACK
		001	NACK
		010	DROP TX
		011	DROP TX CONF
		100	Unused
		101	Unused
		110	
		Flags (B5B6B7B8) (for “Drop TX” and “Drop TX Conf” Control Frames Only)	
		B5	Unused
		B6	Set: New Topology Indication
		B7	Set: More Frames

Field	Size	Bits	Function
		B8	Set: ACK REQ
Sequence Number	TBD1		Cyclic counter. For each source/destination pair, there is one counter for reliable mode and one counter for unreliable mode. Incremented by one for each frame (may be a fragment) sent to a destination.
Port	TBD		To provide port service. Two bits may be enough TBD
TTL	TBD		Optional May be useful for systems carrying audio payload, and for systems with large maximum frame size (where the cost of the extra bits transmitted would not be so high). Used to eliminate outdated frames instead of transmitting persistently to reach final destination. In units of hops or timeslots?
Payload Size	1 Byte		This is used only when the Full Size Frame bit is not set (i.e. only when the payload is not to the maximum value) It's ok to consume one full byte to indicate the size of the payload, since the payload is not fully used anyway. When the "Full Size Frame" bit is set, this field is absent, and the payload size is one byte larger.
Sender	TBD		The address of the node currently sending the frame (it may be the originator or a forwarder)
Next Hop	TBD		The address of the "Next Hop" to which the frame is sent. (decided by the routing protocol)

Table 1 - CRCTP Fields Details

[1] It must be big enough to have one number per in-transit packet. It can be made smaller at the cost of more positive ACK.

2.4 Sequence Number

At each node, a sequence number is increased by one for each generated fragment or frame sent to a destination (sequence number makes no distinction between frames and fragments). Sequence number is not increased for forwarded frame, since sequence number applies to source/destination pair. Also please note that we do not include the port in the algorithm to increase the sequence number. Therefore, two packets with different ports but with the same source & destination address share the same sequence counter.[2]

All frames to a given destination address sent in reliable mode share the same sequence number counter. Similarly, all frames to a given destination address sent in unreliable mode share the same sequence number counter. Therefore, two sequence numbers per destination address are maintained, at each node.

When a sequence number reaches its maximum value, it goes back to zero (it's a cyclic counter). In reliable mode, a frame sent with a sequence number set to the maximum value is also sent with the "ACK REQ" bit set.

[2] It is important to point out here that all application layer flows or packets that shares the same Source & Destination address will share the same Sequence Number in the transport protocol (actually they will share two sequence numbers: one for reliable mode, and one for unreliable mode).
This is done to increase the likelihood of error detection with the NACK mechanism. The NACK mechanism works optimally when frames are regrouped.
Therefore, CRCTP aggregates together all flows and packets from the application layer (with the same source and destination address) for the purpose obtaining reliability at a lower cost. At the destination, together with the re-assembly mechanism, flows and packets will be reconstructed again before being sent to the higher layers.

2.5 Addressing & Forwarding

Although addressing and forwarding are not strictly transport protocol features, they are described here. Because of the way CRCTP works, forwarding is handled in cooperation with the transport protocol. The reason for this is that the protocol must do in-order forwarding, and missed frames are detected by the transport protocol. Also, reliability is handled hop-by-hop; therefore the transport protocol has to inspect frames at each hop to perform its task.

Source and destination address fields are populated by the node that generates the packet. They remain unchanged as packets travel from node to node until they reach their final destination(s). These fields are populated in the frames by the transport protocol; the content of the destination address is based on what the application is requesting. All frames built from the same packet will use the same source and destination address, obviously.

The forwarding engine is responsible for populating the next hop address field, frame by frame. It does so in the same manner whether the frame is locally generated or not (i.e. whether it comes from the higher or lower layers). The forwarding engine asks the routing protocol what is the next hop address, based on the destination address. The forwarding engine populates the next hop address field based on the answer from the routing protocol. The forwarding engine must keep track of the last used “next hop address” to a given destination address. Once it detects a change, it must perform extra steps as described in section 2.11 (

Dynamic Network Topology Issues).

Frames coming from lower layers for forwarding are inspected by CRCTP. As a result of the frame inspection, the transport protocol may send a control message (ACK, NACK, REFUSE) before forwarding the frame. It may also delay the forwarding to a later time due detected missed frames. The reason for this is that the protocol does in-sequence forwarding. If the local node is the (or a) destination, the frame is passed to higher layers or buffered for re-assembly as required. All frames forwarded are also buffered until the node is guaranteed that no retransmission will be requested for this frame.

Performing in-sequence forwarding has the advantage of minimizing overall network usage and minimizing memory consumption especially when considering re-assembly mechanisms. If a node was to send a frame out-of-sequence, it is guaranteed to receive a NACK from the receiver. This could be worked around by having the sender tell the receiver to temporarily accept out-of-sequence forwarding (i.e. do not request a retransmission immediately after detecting a missed frame). This is not allowed in the current design but may be explored in the future (however, in the current design, NACK are sent after waiting some reasonable time to accommodate MAC layer recovery procedure, and network turnaround time).

2.6 Port Service

When an application wants to send to an application on another node, it opens a dialogue with the presentation layer. This is where the port number is determined. The “Port” field is populated by CRCTP in the frame. At the receiver, the port number is used to determine to which application packets should be directed. All frames resulting from a single packets use the same port number. If port service is already implemented elsewhere in the communications stack, CRCTP only needs to retain the appropriate frame fields.

2.7 Reliability

CRCTP provides hop-by-hop reliability through a mix of positive and negative acknowledgements. It mixes the two methods to achieve the low overhead of negative acknowledgement, while maintaining the guaranteed reliability and low memory requirements brought by positive acknowledgment. We believe this is an appropriate compromise between the constraints and requirements of some sensor networks.

2.7.1 The Negative Acknowledgement Method

With the negative acknowledgement method, missed frames are detected by the receiver by looking at a sequence number. If it detects a gap in sequence number, it means that one or more fragments have not been received. When this happens, a request for retransmission of the missed frames is sent to the previous hop through a NACK. This method may also be called passive NACK, but is simply called NACK in this document.

The advantage of this method is that overhead occurs only when recovery is required.

There are several disadvantages to this method:

- A single frame error cannot be detected. We call this “Problem P1”.
- The situation where a node is sending frames to a node that has moved away or is unresponsive is not handled properly. In this case the receiver node will never send a NACK, and the sender will assume the receiver received everything correctly, when this is obviously not the case. We call this “Problem P2”.
- The time it takes to recover from an error is unknown: it may be short if only one frame is lost, but it may be long if many frames are lost before a receiver detects a gap in sequence number.
- The sender node must keep in memory everything it sent to handle eventual request for retransmissions. Since it never receives positive confirmation of reception, it has to guess when history frames can be deleted from memory. This leads to a protocol that requires large memory size to execute. We call this Problem P3”.

Therefore, this method by itself does not work well in networks where small packets occur often or on networks with high mobility or on networks where there is a great variation in link quality.

2.7.2 The Positive Acknowledgement Method

With positive acknowledgment, errors are detected when the sender does not receive the confirmation message from the receiver that it received a frame.

This method has the following advantages:

- Errors are always rapidly detected.
- The sender may delete from memory frames for which the successful reception is confirmed. This leads to low memory requirements.
- It handles correctly both small single frames and long streams.

The main disadvantage of this method is that overhead (ACK) occurs for every successful frame, resulting in high overhead. We call this "Problem P4".

2.7.3 Hybrid Positive-Negative Acknowledgement Method

The method we propose uses both positive and negative acknowledgments together. It keeps a great part of their respective advantages while minimizing their disadvantages. We will explain here how it works and how problems P1 to P4 are eliminated or minimized.

<u>The NACK portion of the hybrid approach</u>

Errors are normally detected using the usual negative acknowledgment method, i.e. by inspecting frame sequence numbers to detect missed frames. Whenever one or more missed frames or fragments are detected, a node starts the $T_{BeforeNACK}$[3] timer (if it was not previously started). If it does receive one missed frame before the timer reaches $T_{BeforeNACK}$, then it resets the timer to zero and updates its list of missed frames. It repeats this until $T_{BeforeNACK}$ is reached. If the timer ever reaches $T_{BeforeNACK}$, while there are still missed fragments, a NACK is sent. The format of a NACK is discussed later.

If a node has detected a missed frame and is waiting $T_{BeforeNACK}$ before sending a NACK, as described above, it may in the mean time receive other frames from that source/destination pair (that are not in the current list of missed frames), and it may

[3] $T_{BeforeNACK}$ is a timer used to let a detected missed frames arrive by itself. For example, a MAC layer may receive multiples out-of-sequence frames in burst mode from a node. Also, some MAC layers may have built-in retransmission mechanism. These mechanisms must have enough time to complete before a transport layer recovery is attempted through a NACK.

or may not realize that it is missing more frames. In this case, it does not reset the $T_{BeforeNACK}$ timer and it continues the procedure as described in the previous paragraph, but with the updated list of missed frames. At any node, there can be at most one $T_{BeforeNACK}$ timer per source/destination pair.

When a node sends a NACK, it request a retransmission for all missed frames that have been reported missing, for the given source/destination pair, **before the $T_{BeforeNACK}$ timer was last reset.** This is to avoid requesting retransmission for missed frames that may still be in transit. After sending a NACK, if there are still missed frames for which no retransmission was requested, the $T_{BeforeNACK}$ timer is reset; otherwise, the timer can be stopped (deleted). Doing a reset of the timer at this point ensures that CRCTP does not request missed frames one by one, in the event that they occur regularly and periodically.

After sending a NACK, the sender of the NACK waits for a period of T_{NACK}. If it did not receive any requested missed frame within that period, is sends a NACK again, with the updated list of missed frames, as long as the list of frames ready to be requested is not empty. A node may request more than one missed fragment in a single NACK. Every time a missed fragment is received (i.e. recovered), T_{NACK} is reset. If T_{NACK} is reached, another NACK is sent. A retransmission request for a given frame must not be tried more than N_{NACK} times. Therefore, for each frame a retransmission count is kept in memory to ensure that a given frame is not requested more than N_{NACK} times. Frames that have been requested more than N_{NACK} times are flagged as such and are no longer requested in NACKs.

Therefore, NACKs are sent whenever $T_{BeforeNACK}$ or T_{NACK} expires. Whenever this happens, a new NACK is generated, requesting a retransmission for all missed frames for which it is time to request a retransmission.

A great advantage of this approach compared to traditional NACK approaches is that normally sequence numbers are used only for fragments of a packet, or for "stream" type of flows[4]. This limits this overhead-efficient method to only this type of traffic. We propose to extend the method to any outgoing frame with the same source-destination address. Therefore, the sequence number is increased for each

[4] Therefore, application layer packets and flows are aggregated into one single transport layer sequence of frames. Only one sequence number is shared for all application layer traffic, and forwarder nodes allocate memory for all active "Source/Destination" pair they encounter. Forwarder nodes may release this memory when the "Source/Destination" pair is no longer active on this forwarder. Application layer packets and flows are re-constructed with the re-assembly mechanism. This process is independent for reliable and unreliable traffic.

outgoing frame from a node to a same destination. The only requirement for the proposed method to function is to be confident that there are more frames to be transmitted (frames with the same source/destination pair). This is determined by the source node through an examination of the frames/fragments awaiting transmission. If there are such frames, this is indicated by setting the "More Frames" flag. The receiver therefore knows that more frames are expected to arrive from that source (this information may be used to react to topology changes, see section 2.11). When the source node detects that no more frames are ready or expected to be sent the destination, it sets the "More Frames" flag to zero and set the "ACK REQ" bit to one. The receiver will then positively acknowledge this last frame, and the sender and receiver can clear their buffer for these frames.

The negative acknowledgement approach is the most efficient approach to provide reliability in term of overhead. With our approach, the negative acknowledgement method is used potentially more often that with traditional NACK protocols. This technique greatly minimizes the overhead associated with positive acknowledgement, that we called problem P4 earlier.

The ACK portion of the hybrid approach

With CRCTP, at specific times, a sender node requests from the receiver a positive acknowledgement of a frame. This is necessary to overcome the deficiencies of the pure negative acknowledgement approach. Positive acknowledgments are requested by setting the "ACK REQ" bit to one. A sender must set the "ACK REQ" bit on the following occasions:

A. When the "More Frames" bit is not set
B. After sending N_{FrACK} frames (with the same source/destination address) without the "ACK bit" set

Or, whichever occurs first:

C. When a period of T_{FrACK} elapsed since the last frame with the same source/destination address and with the "ACK REQ" bit set was sent
D. When a sender (originator node or forwarder node) must release some memory

A, B and C above effectively solves Problem P1 and P2 (about missed frames never detected) mentioned earlier. B, C and D effectively solve the memory problem P3 discussed earlier.

When a receiver receives a frame with the ACK REQ bit set, it first verifies if it is missing any frames by inspecting the sequence numbers. If it is missing any frame, it sends a NACK requesting all of them (regardless of the value of the $T_{BeforeNACK}$ or T_{NACK} timers). When all frames are received up to the one that was received with the ACK REQ bit set, the receiver node must send an ACK frame. ACK frame format is discussed later in this document.

After requesting an ACK, a sender starts a timer called T_{ACK}. The T_{ACK} timer is reset to zero upon reception of a NACK (for frames with the same source/destination address and with a sequence number less than or equal to the one of the frame for which an ACK as requested). The T_{ACK} timer is also reset every time the ACK sender does a retransmission of frame whose sequence number is lower than or equal to the one of the frame for which an ACK as requested. The T_{ACK} timer continues its count/reset procedure until the ACK is finally received. If the T_{ACK} timer reaches T_{ACK} and has never received the ACK, the frame with the "ACK REQ" bit set is sent again, and the timer is reset. The sender node may try this for a maximum of N_{ACK} times, after which it gives up.

This method of requesting a positive acknowledgment once in a while enables nodes to reduce and control their memory consumption. After reception of an ACK, a node is guaranteed that the next hop receiver received all previous frames with the same source/destination addresses. Therefore, it no longer needs to store these frames in memory. Moreover, adjusting N_{FrACK} and T_{FrACK} has the effect of giving a boundary to the maximum per/hop worst case frame latency. These advantages come at the cost of the overhead of transmitting ACK frames once in a while.

The effect of mixing the two approaches

As explained before, with the NACK approach, no overhead occurs until errors are detected. Therefore, the performance of such a method approaches that of a circuit switched network when there are few errors. However, it may lead to high memory consumption due to potential long error discovery time, plus it can't provide guaranteed reliability for small packets. On the other hand, pure ACK results in low memory consumption due to rapid error recovery.

Mixing the two approaches allows us to consume a certain controllable amount of memory, to improve network performance in term of control frame overhead. Worded differently, it allows us to consume a certain amount of network resources to save on device memory resources. This mix is configurable, and potentially dynamically configurable. Also, because the approach is hop-by-hop and not end-to-end, error recovery is done more rapidly and consumes much less network resources. Mixing ACK and NACK also has the advantageous inherent effect of distributing memory consumption among the forwarder nodes between the source and the destination (in the case of large packets, streams or continuous frame transmission).

2.7.4 Proactive Negative Acknowledgement

As explained before, a node sends a negative acknowledgement when it detects a missed frame by inspection of the "Sequence Number". We call this the passive negative acknowledgement.

A proactive negative acknowledgement is one where a receiver knows that there are more frames to come (in the case of our protocol, by inspecting the “More Frames” bit), but they don’t receive any after some long delay, and there has been no missed frames detected. Therefore, under the rules of the passive negative acknowledgement, a NACK is not allowed to be sent in this case. A proactive NACK could be sent in this case. However, with CRCTP, this is not necessary since the sender requests positive ACK periodically.

Therefore, proactive NACK is not allowed with CRCTP.

The “More Frames” bit is used for other purpose in CRCTP (see section 2.11).

2.8 Un-Reliability

Sending in unreliable mode is possible with our proposed protocol. The frame format is the same, but meaning of the flags is different than for reliable frames (see Table 1). The “ACK REQ” and “More Frame” bits are not used.

In the unreliable mode, it is not necessary to inspect the “Sequence Number” to detect missed frames or fragments. It is only necessary to inspect them for packet re-assembly purpose.

Finally, in the unreliable mode, receiver and forwarder nodes never send ACK or NACK. Forwarders never buffer any frame or fragment – they just pass them trough. Receivers (final destination only) use buffering of frames only if these are fragments of packets, for re-assembly purpose.

2.9 Fragmentation

Locally generated packets enter the transport protocol from higher layers. If a packet, including transport header and footer, fits into one single MAC frame, it does not have to be fragmented. Otherwise, the transport protocol fragments the packet into multiple fragments. Because the protocol does in-sequence forwarding, fragments of a packet are sent in sequence. Fragments from packets coming different ports can be interlaced (but they share the same sequence number). The reassembly mechanism will perform reassembly on a source/destination address-port number basis.

The first fragment of a packet is sent with the “First Fragment” bit set; this bit is set to zero for all other frames. The last fragment is sent with the “Last Fragment” bit set; this bit is set to zero for all other frames. In unreliable mode, all other fragments of a packet are sent with the “A Fragment” bit set (this bit does not exist in reliable mode). Frames of packets that are not fragmented are sent with these bits set to zero.

2.10 Reassembly

The reassembly process takes place at the destination only. It does not take place at the forwarder nodes, unless a forwarder is also a destination (in the case of broadcast or multicast). The reassembly process is different for reliable and unreliable mode. In both modes, a node knows that it is receiving one or more fragmented packets by seeing at least one frame received with the "First Fragment", "A Fragment" or "Last Fragment" bit. In all cases, it starts to buffer these fragments for re-assembly purpose.

Formally, a reassembly process is started as soon as all the following conditions are met:

1. The transport protocol receives a frame with the "First Fragment", "A Fragment" or "Last Fragment bit set.
2. The destination address is the local node (this frame is now at its intended destination)
3. A reassembly process was not already started for this Source/Destination/Port combination.

Once a packet is re-assembled, it no longer needs to be buffered for re-assembly purpose (but may still need to be buffered for reliability reasons, see 2.7)

One could argue that in reliable mode, if a node receives the "Last Fragment" first, it may not have buffered the previous, but not first, fragments because the "A Fragment" bit is not used in reliable mode. However, missed frames are buffered at least until the First Fragment is received, since the protocol does in sequence forwarding (see 2.7).

In reliable mode, at the final destination, a packet is immediately re-assembled once the "First Fragment", "Last Fragment" (with the same source/destination/port) AND all fragments in between (with or without the same port) have been successfully received. Note that this does not prevent multiple packets to be interlaced together in transport frames. However, there may be some situations where all frames of an application layer packet are correctly received but cannot be re-assembled until some frames of another packet are successfully received (because they share the same sequence number).

In unreliable mode, the situation is a little bit more complicated. Whenever a fragment of a new packet is received at a final destination in unreliable mode, a reassembly timer $T_{\text{Reassembly}}$ is started (if it's a newly encountered packet) or reset (in the case where this is not the first fragment of the packet). When the last fragment of a packet is received in unreliable mode, the packet is immediately re-assembled if all fragments were received. If some fragments were not received, the reassembly timer is reset. If the reassembly timer ever reaches $T_{Reassembly}$, then the associated packet is re-assembled, even if some frames are missing. Known missed fragments are zero padded in the re-constructed packet. If the last fragment was never received, the frame with the latest sequence number is treated as the last one. If the first fragment was never received, the fragment with the “A fragment” bit set that has the lowest sequence number is treated as the first one. In all cases where a packet is reassembled but with missed fragments, the re-assembled packet is sent to higher layers with an indication of erroneous packet. We note that with this algorithm, there is a possibility that, if a last fragment is never received and the first fragment of the next packet is also never received, these two consecutive packets get treated as only one, and it is erroneous. However, this is in the unreliable mode only, and in this case the protocol operates in “best effort” mode anyway. For specific implementation, the designer may want to implement a checksum or a cyclic redundancy check to detect such erroneous frames. This is not specified in the base specification of CRCTP because it was considered that the extra overhead was not necessary, given that this is for unreliable mode only.

Therefore by using this scheme, in sequence reception is not required for either frames (fragments) or packets. Also, multiple fragmented packets can be received concurrently from the same source without confusion.

2.11 Dynamic Network Topology Issues

An important situation to consider is when the forwarder nodes are changed during the course of a transmission. Let's consider the following situation. A source node A is sending to node D. In the beginning of the transmission, B acts as relay between A and D. Later on, this is changed and C becomes the relay. This is illustrated in Figure 6.

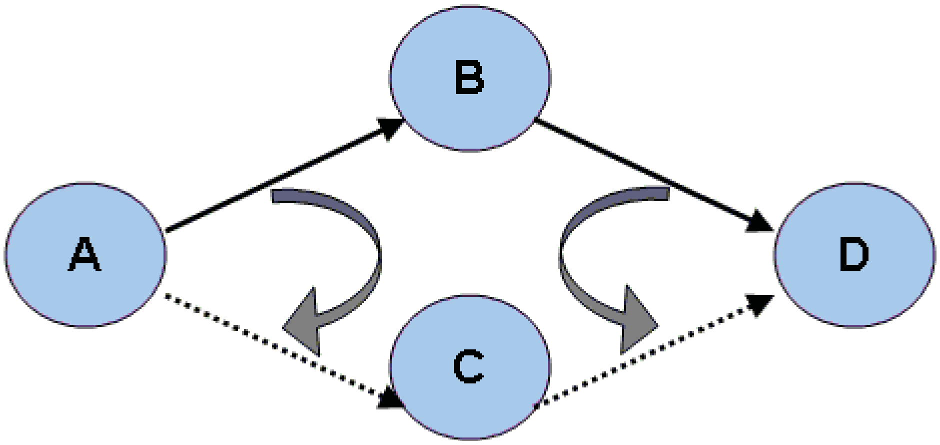


Figure 6 - Topology Change

There are two issues to consider.

1. Node C becomes involved in the transmission sometimes after it has started but before it ended. The first frame it sees is <u>not</u> the first frame or fragment. Normally, according to NACK rules, it would request the missing previous frames, but it should not in this special case. But how does it know that it is a new relay for an ongoing transmission? It may think that it missed all previous frames and has always been the selected relay.
2. Node B stops its involvement in the transmission. How does it know it is no longer responsible for this flow? How does it know that that it is acceptable to not receive a requested positive ACK from the next hop receiver? How does it know that it can drop the buffers for that transmission?

Both of these issues have to be addressed. The first node to be aware of the changing situation is node A. When node A requests the "Next Hop" address from the forwarder, it monitors for a change from the previously used "Next Hop address". If a change is detected, the following actions are taken:

1. Node A informs B that it is no longer involved in this transmission by sending it a "DROP TX" frame. Note that B may no longer be a

neighbor (therefore the "DROP_TX" is treated like a data packet and may be routed). The format of the "DROP TX" frame is detailed later.

- On reception of this "DROP TX" frame, node B replies with a DROP TX CONF" frame.
- The transmission of the actual data frame on the new route can take place right after the first "DROP TX" has been sent. It is not necessary to wait for reception of the "DROP TX CONF" frame.
- The "DROP TX" and "DROP TX CONF" are control frames, but are treated like reliable data frames (they are routed). Therefore, it can be assumed that they will successfully reach their destination.

2. Node A informs node C that it is a new forwarder of an ongoing transmission, by sending (to node C) the next frame with the "New Topology Indication" bit set. In this case the "ACK REQ" bit is also set.
   - Node C immediately forwards the frame to node D, with the "ACK REQ" bit set. It waits for the positive ACK from D before sending its own positive ACK to A. To allow enough time for this extra exchange to take place, node A waits longer than it normally does for the reception of the ACK from C. It should wait $T_{ACKTopologyChange}$.
   - If node C instead receives a NACK from D, it cannot handle it because it does not have yet any frames buffered for that transmission. In this case, instead of sending an ACK to node A, it sends an NACK requesting the missing frames. After this, normal procedures commence.
   - If node C hears a NACK from node D to node B for the source/destination flow for which it is now responsible, it must respond to it (with the mechanism described in the previous paragraph). This may happen when node D is sending a NACK before it has realized that the new previous hop is node C.

Node D observes the change by inspecting the address of the previous hop. It always uses the "last" previous hop observed to send control messages (ACK, NACK).

When node B becomes aware that it is no longer a forwarder for the transmission, it can drop any ongoing procedure or buffer related to that transmission.

Step 1 above works well in networks where routes are rapidly constructed or proactively maintained, such as the TinyOS implementation of sensor network by RASN. In the case of the Newtrax nodes, having node A send a message (the DROP_TX) to node B when the route between them has to be reconstructed may not be a good idea since the new route may take long to be alive. It is suggested to use the following alternative approach instead:

1. Node B knows that a transmission is not over when the last frame was received with the "More Frames" bit set.

- If it did not hear any activity on this transmission for a period of $T_{TopologyChange}$, it can conclude that it is no longer involved in that transmission and it may release its buffer for all concerned frames. It may also give up any process related to this transmission (ACK, NACK).
- Node B does not necessarily have to abandon the transmission right at $T_{TopologyChange}$ of inactivity, but it can abandon it from this point on. Abandoned transmissions may be checked for periodically, or when memory becomes low.

In conclusion, when a relay node is modified on the fly, the overhead penalty is two control messages being sent: “DROP TX” and “DROP TX ACK”. Another penalty that may occur is a longer recovery time of frame lost, because the new forwarder may not have them in memory: the recovery has to go one hop further. As for the flow itself, it can continue flowing on the new route almost immediately.

In the case where the alternate solution is implemented instead, the penalty is (instead of the overhead of “DROP TX” & “DROP TX CONF”) the overhead of node B continuing its ACK and NACK procedures related to the transmissions in which it is no longer involved (these procedures are abandoned after a while by node B, after the maximum number of attempts).

2.12 Broadcast – Multicast

The capability of CRCTP to provide reliable transport for broadcast and multicast depends on the cooperation that it can achieve with the routing protocol or the MAC layer. We will therefore consider two scenarios. The first scenario is when CRCTP can learn from the routing protocol or the MAC layer who the next hop receivers of multicast or broadcast frames are. The second scenario is when it cannot obtain this information (independent operation of the transport protocol).

Cooperation with the routing protocol is highly desirable. Without it the design is not optimal.

In the case where broadcast/multicast is not supported by the routing protocol or by an independent multicast protocol, CRCTP implements a simple pure flooding protocol, implemented inside the forwarder.

We note that multicast and broadcast frames are identified by inspecting the destination address. Multicast and broadcast addresses must be identifiable.

2.12.1 Scenario 1 – Cooperation with the Routing or MAC Layer (Recommended)

In this scenario, we assume that CRCTP knows who the next hop receivers of broadcast/multicast frames are. This information may be given by the routing protocol or by the MAC layer, depending on how the networking stack is designed.

When CRCTP sends a broadcast/multicast frame, it does so with the same mechanism as described for unicast frames, in term of reliability mechanism (ACK, NACK, sequence numbers, flags, etc). However, there are two differences.

The first difference is when the forwarder receives a frame from the lower layers, the decision to forward it or not is left to the routing protocol if it has built-in support for multicast/broadcast or to an external multicast protocol. If the routing protocol does not support it and there is no external multicast protocol, then the frame is forwarded once only, and that decision is taken by the forwarder itself (in other words, the forwarder supports flooding, but if the routing protocol or multicast protocol has a better mechanism, it may be used instead).

The second difference is that after sending a multicast/broadcast frame with the "ACK REQ" bit set, the sender expects to receive an ACK from each of its neighbours. There is only one T_{ACK} timer started, and all ACKs have to be received before its expiry to avoid a retry, as per the unicast mechanism explained earlier in this document. Actually, the mechanism for waiting for an ACK is the same as for unicast except that it waits for more than one ACK. If one or more ACKs never arrive, the frame is sent again as per the unicast rules (for example, if three ACKs never arrive, the frame is retransmitted only once on this attempt, not three times, obviously). We assume that the MAC layer resolves the collisions between the many ACKs that are generated at the same time. We note that the configuration of the ratio between ACK/NACK is important especially when considering broadcast/multicast, because the cost in overhead of requesting a positive ACK is more important compared to unicast.

Receivers of broadcast/multicast frames behave just the same as for unicast. They therefore send ACK and NACK as described in the unicast section of this document. The only difference is that a receiver may also be a forwarder, in which case they also act as forwarder as described earlier in this section.

2.12.2 A Note on Flooding

As mentioned in the previous section, if there is no multicast protocol and if the routing protocol does not directly support multicasting, then the forwarder is capable of falling back to a pure flooding mechanism.

In this case, every broadcast/multicast frame is normally rebroadcasted only once. This means that a node must keep track of the broadcast/multicast frames it sent in order not to send them more than once (the exception if for retransmission requested by NACKs). Because CRCTP does in-sequence forwarding, it is very easy to know if a frame was already transmitted or not. CRCTP only needs to keep in memory the sequence number of the last broadcast/multicast frame sent. If a frame arrives and its sequence number is lower than the last one sent, then CRCTP is sure that this frame has been already transmitted at least once. Therefore, in term of extra memory consumption to support this flooding mechanism, CRCTP only needs to keep in memory the sequence number of the last frame sent for every source/destination pair of multicast or broadcast frames. CRCTP can release the "last sequence number" in memory for a source/destination pair when a frame arrives with the "More Frames" bit not set. We can therefore conclude that the added memory consumption to support flooding for reliable traffic is minimal; most of the memory consumption is related to providing reliability and re-assembly.

2.12.3 A Note on Unreliable Broadcast/Multicast

As mentioned in the previous section, if there is no multicast protocol and if the routing protocol does not directly support multicasting, then the forwarder is capable of falling back to a pure flooding mechanism.

Akin unicast, with broadcast and multicast in unreliable mode, frames are just forwarded as they arrive, and none of the reliability mechanisms (ACK, NACK) operates.

If there is a multicast protocol, or if the routing protocol supports it, the decision to forward or a not an unreliable broadcast/multicast frame remains to those entities. If there is no such entity, then CRCTP's forwarder is responsible for performing pure flooding. Therefore, every frame is re-broadcasted only once. Because in the case of unreliable mode the forwarding is not necessarily done in-sequence, CRCTP must keep track of all multicast/broadcast frames transmitted for each pair of source/destination it encounters. This is done by keeping a vector of sequence numbers that were already transmitted for each pair of source/destination. However, keeping in memory the sequence number of every multicast/broadcast frames sent in unreliable mode results in constantly increasing memory consumption. To avoid problems of memory, the following techniques are used:

1. Sequence numbers of frames older than $T_{Reassembly}$ <u>can</u> be deleted from memory. This means that these sequence numbers may be deleted if they are older than $T_{Reassembly}$, but it is not necessary to do it as soon as $T_{Reassembly}$ is reached. Memory "cleaning" can be done periodically or as required.
2. Memory space can be saved by using the same technique as for representing missed fragments in NACK (i.e. out of sequence "Sequence Number" means a range of frames) (see 2.14.2).
3. It is assumed that all frames lower than the lowest stored sequence numbers have been transmitted once

Other than that, operation of unreliable mode is the same as for unicast. We refer to section 2.8 for operation of fragmentation/reassembly of multicast/broadcast frames, since it is the same as for unicast.

2.12.4 Scenario 2 – Independent Operation

If cooperation with other layers is impossible to learn the next hop receivers of broadcast/multicast frames, the operation is very similar but the difference is that positive ACK are never requested (the "ACK REQ bit is never set). Only the mechanism with NACK is enabled.

It should be noted that this scenario is not optimal. 100% reliability cannot be guaranteed, especially for small packets. The bigger the packets are, the better the reliability becomes. This is a fallback mechanism that should be implemented only when cooperation with other layers is impossible.

Please note that this is the only option; there are ways to make it better. A possibility is to request ACK for the first few frames of a packet. After receiving

ACKs for a certain number of frames, a node can have a good idea of who its neighbors are. More frames sent with ACK requested actually means a greater probability of having the correct picture of the neighborhood. However, this becomes probabilistic reliability, not 100% reliability, and it still does not cope well with small frames and highly dynamic topology. An independent neighbor discovery could also be implemented at the transport layer, however this is grossly inefficient.

Finally, if an independent operation must be implemented, proactive NACKs should be implemented, by having receiver nodes inspect the "More Frames" bit and by having a timer triggers a proactive NACK when it expires.

To conclude, if we must fall back to this non optimal approach, this section of the document will be revised so the mechanisms are described more formally.

2.13 Scheduler

As seen in Figure 3, the transport protocol does not communicate directly with the data link layer: frames must pass through a scheduler. In term of real implementation, this does not need not to be a separate layer. It could be implemented within the transport protocol itself, or with the appropriate forwarding layer of a communications stack.

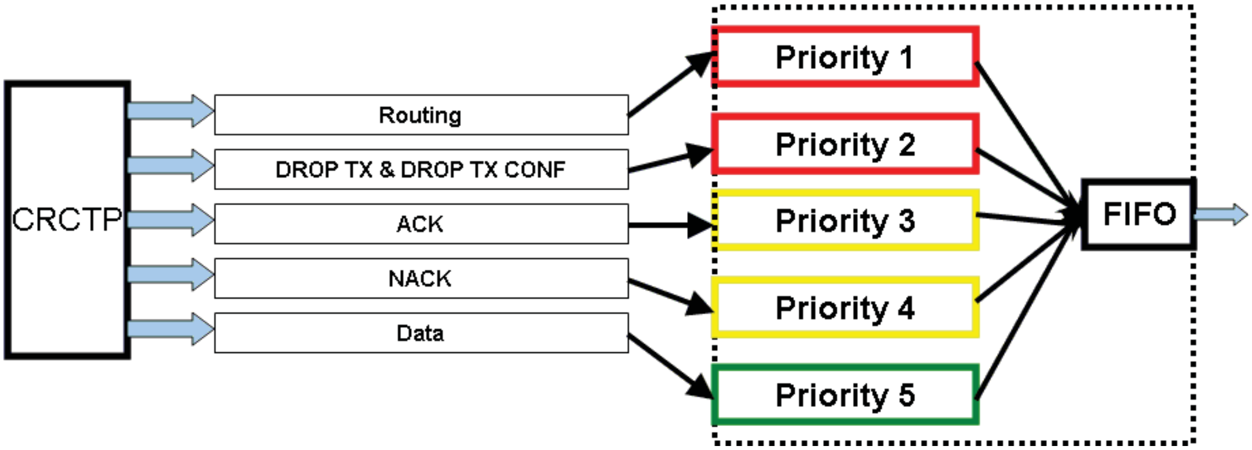


Figure 7 - CRCTP Scheduler

The scheduler, as required by this protocol, is simple. It is illustrated in Figure 7. It is meant only to give higher priority to control frames over data frames. The actual priority is the following (one being the highest).

1. Routing
2. “DROP TX” & “DROP TX CONF”
3. ACK
4. NACK
5. Data

The highest priority is given to routing traffic, since the transport protocol rely on it for its operation. Even though routing may be considered as data from the view of the transport protocol, it should be distinguished for scheduling purpose (in other word, routing traffic may use reliable or unreliable mode, but these data frames should be identified so they are given high priority).

The next highest priority is given to “DROP TX” and “DROP TX CONF” since receiving them in time could potentially save network resources.

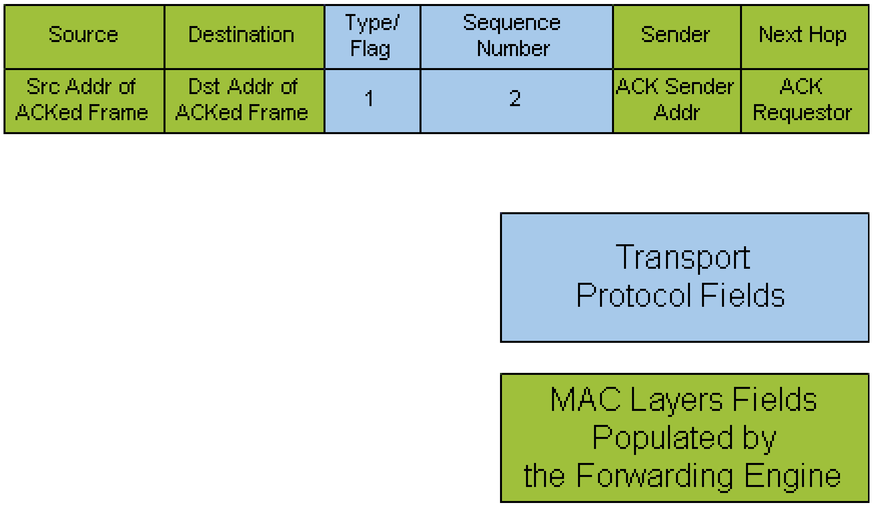
ACK and NACK are given higher priority than data since their role is to ensure proper data transport. ACK is given a higher priority than NACK since its successful delivery may result in a node being able to release precious memory that it needs for other frames or flows.

The scheduler de-queues frames with a pure priority scheme. Eventual refinements could be brought to the scheduler, where data could be further categorized and prioritized. Also, scheduling other than pure priority may be considered for the different categories data frames.

2.14 Control Frame Format

In this section, the frame format of the control frames is illustrated and described.

2.14.1 ACK Frame Format



Source	Destination	Type/ Flag	Sequence Number	Sender	Next Hop
Src Addr of ACKed Frame	Dst Addr of ACKed Frame	1	2	ACK Sender Addr	ACK Requestor

Transport Protocol Fields
MAC Layers Fields Populated by the Forwarding Engine

Figure 8 - ACK Frame Format

We note that the ACK frame has no port field. This is because the port field is not necessary to uniquely identify fragments and frames, since frames coming from all applications, regardless of their port numbers, share the same sequence number in CRCTP (unlike UDP and TCP). Furthermore, a single ACK can be used to acknowledge frames with different port numbers.

Field	**Size**	**Bits**	**Function**
Source	TBD		This is the source address of the frame being acknowledged
Destination	TBD		This is the final destination address of the frame being acknowledged
Type/ Mode/ Flags	1 Byte	Type (B1)	
		Set to 1	To indicate a Control Frame
		Control Frame Sub-Type (B2B3B4)	
		Set to 010	To indicate a “DROP TX” Frame
		B5	Unused
		B6	Unused
		B7	Unused
		B8	Unused

Field	Size	Bits	Function
Sequence Number	TBD		Same as for a data packet (this frame is treated as a data packet with the ACK REQ bit set)
Sender	TBD		This is the address of the node that is generating the ACK
Next Hop	TBD		This is the address of the node that requested the ACK

Table 2 - Description of the Fields of ACK Frames

2.14.2 NACK Frame Format

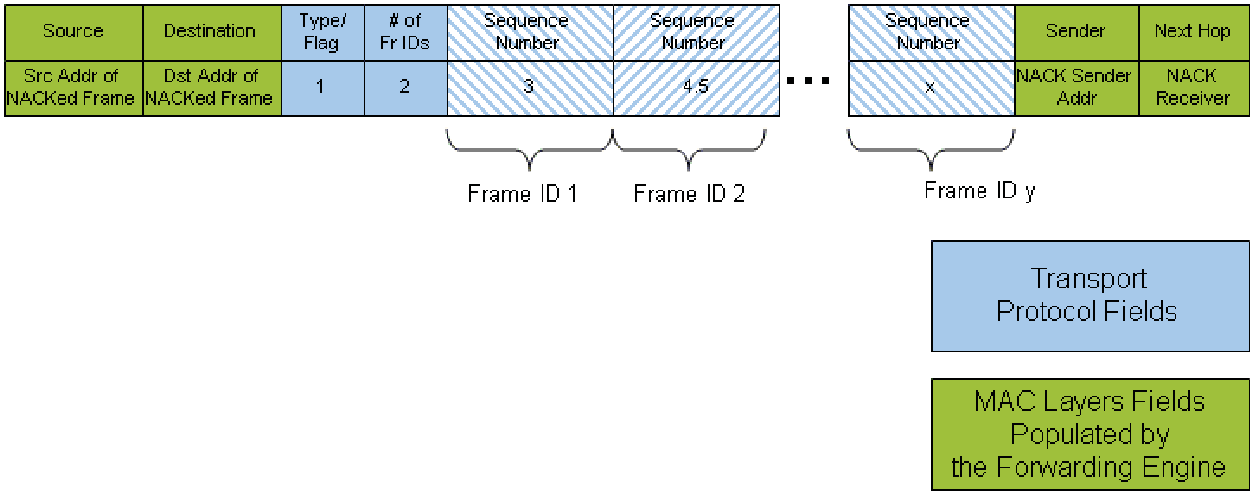


Figure 9 - NACK Frame Format

It is important to note that more than multiple frame retransmission can be requested in a single NACK frame. It is also possible to request retransmission for range(s) of frames.

The frames for which a retransmission is requested are uniquely identified in the NACK packet by their “Sequence Number”. The frames for which a retransmission is requested are listed in order in the NACK frame.

When a retransmission for range of frames is requested, this is done by inserting the boundaries of the range in the reverse order.

For example, let’s say we have frames 1 to 10. The receiver node discovers that it is missing frames 2, 5, 6, 7 and 9. Then, it lists the frame IDs in the following order in the NACK frame: “2, 7, 5, 9”. The receiver of the NACK frame therefore knows that the pair “7, 5” actually represent a range (5 to 7) because it is in reverse order.

We note that the NACK frame has no port field. This is because the port field is not necessary to uniquely identify fragments and frames, since frames coming from all applications, regardless of their port numbers, share the same sequence number in CRCTP (unlike UDP and TCP). Furthermore, a single NACK can be used to report missing frames that have different port numbers.

Field	**Size**	**Bits**	**Function**
Source	TBD		This is the source address of the frame(s) for which a NACK is sent (i.e. source address of the frame(s) for which a retransmission is requested)
Destination	TBD		This is the final destination address of the frame(s) for which a NACK is sent (i.e. destination address of the frame(s) for which a retransmission is requested)
Type/ Mode/ Flags	1 Byte	Type (B1)	
		Set to 1	To indicate a Control Frame
		Control Frame Sub-Type (B2B3B4)	
		Set to 001	To indicate an ACK Frame
		B5	Unused
		B6	Unused
		B7	Unused
		B8	Unused
# of Frame IDs	1 Byte		A number that indicates the number of frame IDs to follow If NACK is 16 or less, B5B6B7 B8 from the "Type/Mode/Flag" field can be used instead of using a separate field
Sequence Number	TBD		Used to identify a frame. Set to the value of the Sequence Number of the frame for which a retransmission is requested Note: A sequence number, source address and destination address of a frame form its frame ID (with these the frame can be uniquely identified)
Sender	TBD		This is the address of the node that is generating the NACK (i.e. the node that is requesting retransmission(s))
Next Hop	TBD		This is the address of the node to which the NACK is sent (i.e. the node from which we are requesting a retransmission)

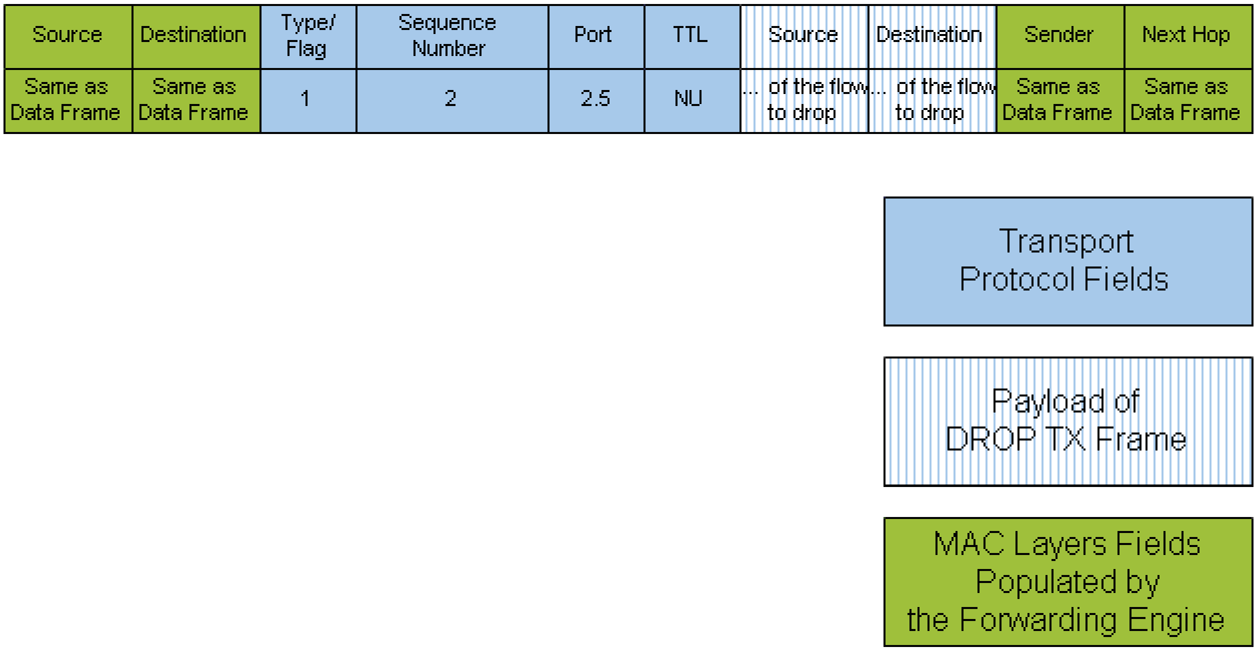
Table 3 - Description of the Fields of NACK Frames

2.14.3 DROP TX Frame Format

A “DROP TX” frame is essentially treated like a data frame (i.e. it is routed). The difference is that its payload carries transport protocol control information, therefore it is called a control frame. This frame is also a little different in its “Type/Frame/Flag” field (it uses a portion of the control and a portion of the data flags). It shares the sequence number with the data frames.

Its payload carries the source and destination address of the transmission to be dropped by the destination of this frame. With the source and destination address of the flow to be dropped, the flow to be dropped is uniquely identifiable.

Please note that in this section we refer to a flow as a sequence of frames that share a common “source/destination”.



Source	Destination	Type/ Flag	Sequence Number	Port	TTL	Source	Destination	Sender	Next Hop
Same as Data Frame	Same as Data Frame	1	2	2.5	NU	... of the flow to drop	... of the flow to drop	Same as Data Frame	Same as Data Frame

Transport Protocol Fields
Payload of DROP TX Frame
MAC Layers Fields Populated by the Forwarding Engine

Figure 10 - DROP TX Frame Format

Field	Size	Bits	Function
Source	TBD		This is the address of the node that generates the “DROP TX” frame. (NB same definition as for data frame)
Destination	TBD		This is the final destination of the “DROP TX” frame. In other words, this is the address of the node that is no longer involved in a transmission, or the “previous forwarder”. (NB Same definition as for a data frame)
Type/	1 Byte	Type (B1)	

Field	Size	Bits	Function
Mode/ Flags		Set to 1	To indicate a Control Frame
		Control Frame Sub-Type (B2B3B4)	
		Set to 010	To indicate an DROP TX Frame
			B5 (MORE FRAME)	Set to 0 or 1 as per rules described in 2.7.3.
		B6 (ACK REQ)	Set to 0 or 1 as per rules described in 2.7.3.
		B7(Full Frame Size)	Set to zero (this frame is never a full size frame, its payload is known and small)
		B8 (New Topology Indication)	Set: New Topology Indication (same rules as for data frame, see 2.11) (even though this frame is to control a topology change, it may serve to indicate another new topology change along its way, just like a data frame could)
Sequence Number	TBD		As if it was a data frame
Port			Zero (unused since this is a control frame)
Source (of the dropped flow)	TBD		The source address of the flow to be dropped.
Destination (of the dropped flow)	TBD		The destination address of the flow to be dropped.
Sender	TBD		As if it was a data frame
Next Hop	TBD		As if it was a data frame

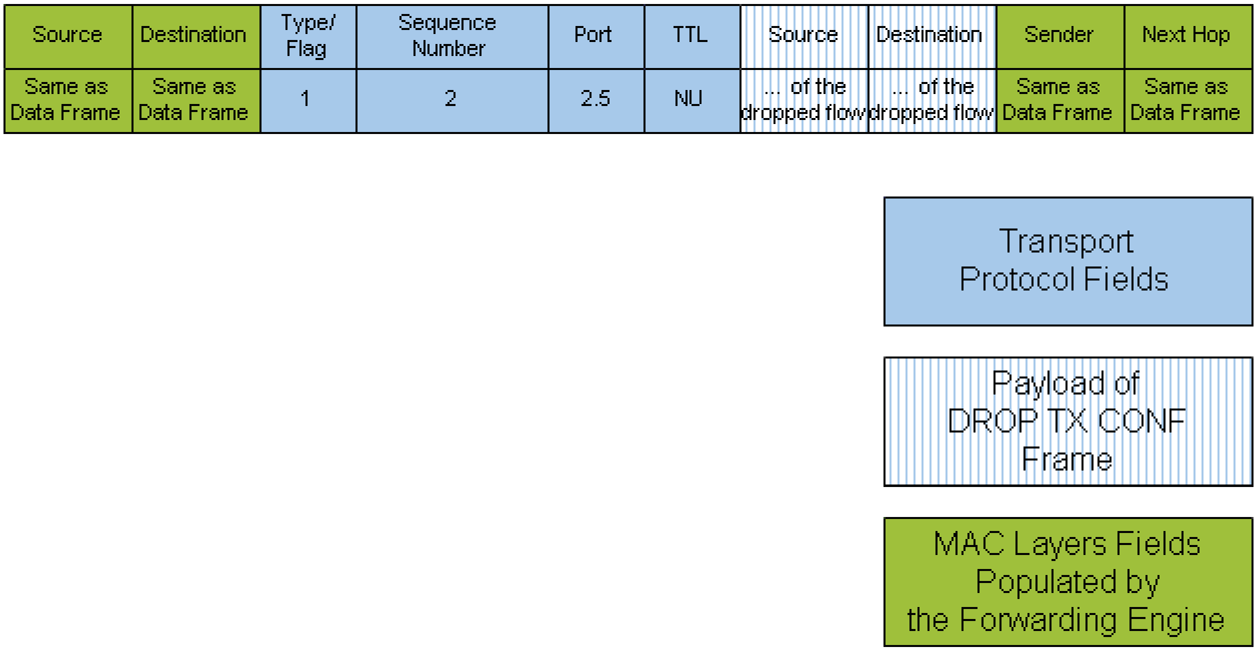
Table 4 - Description of the Fields of "DROP TX" Frames

2.14.4 DROP TX CONF Frame Format

Like a “DROP TX” frame, a “DROP TX CONF” frame is essentially treated like a data frame (and it is routed). Here again, the difference is that its payload carries transport protocol control information, therefore it is called a control frame. This frame is also a little different from the data frame in its “Type/Frame/Flag” field (it uses a portion of the control and a portion of the data flags). It shares the sequence number with the data frames.

Its payload carries the source and destination address of the transmission that was dropped by the node that generates this frame. With the source and destination address of the flow that was dropped, the flow to be dropped is uniquely identifiable.

Please note that in this section we refer to a flow as a sequence of frames that share a common “source/destination”.



Source	Destination	Type/ Flag	Sequence Number	Port	TTL	Source	Destination	Sender	Next Hop
Same as Data Frame	Same as Data Frame	1	2	2.5	NU	... of the dropped flow	... of the dropped flow	Same as Data Frame	Same as Data Frame

🟦 Transport Protocol Fields

▨ Payload of DROP TX CONF Frame

🟩 MAC Layers Fields Populated by the Forwarding Engine

Figure 11 - DROP TX CONF Frame Format

Field	Size	Bits	Function
Source	TBD		This is the address of the node that generates the “DROP TX CONF” frame, i.e. the address of the node that just dropped its role as a forwarder of a flow (NB same definition as for data frame)
Destination	TBD		This is the final destination of the “DROP TX CONF” frame. In other words, this is the address of the node that initially sent the “DROP TX” frame.
Type/ Mode/ Flags	1 Byte	Type (B1)	
		Set to 1	To indicate a Control Frame
		Control Frame Sub-Type (B2B3B4)	
		Set to 011	To indicate an “DROP TX CONF” Frame
		B5 (MORE FRAME)	Set to 0 or 1 as per rules described in 2.7.3.
		B6 (ACK REQ)	Set to 0 or 1 as per rules described in 2.7.3.
		B7(Full Frame Size)	Set to zero (this frame is never a full size frame, its payload is known and small)

Field	**Size**	**Bits**	**Function**
		B8 (New Topology Indication)	Set: New Topology Indication (same rules as for data frame, see 2.11) (even though this frame is to control a topology change, it may serve to indicate another new topology change along its way, just like a data frame could)
Sequence Number	TBD		As if it was a data frame
Port			Zero (unused since this is a control frame)
Source (of the dropped flow)	TBD		The source address of the flow that was dropped.
Destination (of the dropped flow)	TBD		The destination address of the flow that was dropped.
Sender	TBD		As if it was a data frame
Next Hop	TBD		As if it was a data frame

Table 5 - Description of the Fields of "DROP TX CONF" Frames

2.15 Timer & Counter Values

In this section all timers and counters of CRCTP are explained and their recommended values are given.

We define the following values for proper understanding of this section.

$N_{MACRetries}$: The maximum number of attempts of transmission at a MAC layer
T_{MAC}: The maximum time it takes for a MAC layer to transmit a frame once it starts its transmit procedure for a frame (includes RTS, CTS, ACK, interframe space, time waiting to access medium, retransmissions, etc)
$T_{TransData}$: The transmission time of a data frame (related to modulation and frame size and overhead)
$T_{TransNACK}$: The transmission time of a NACK frame (related to modulation and frame size and overhead)
$T_{TransACK}$: The transmission time of an ACK frame (related to modulation and frame size and overhead)
T_{Prop}: The propagation time of the frame, in the air. For small distances, we may assume this to be close to zero.

The timers and counters defined in the following sections are those of CRCTP.

2.15.1 $T_{BeforeNACK}$

When a node detects a missed frame by inspection of the sequence number, it starts the $T_{BeforeNACK}$ timer, if it was not done already. Only when this timer reaches its target value does a node send a NACK. This is necessary to let frames that may still be in transit arrive out-of-sequence. Even though CRCTP does in-sequence forwarding, out-of-sequence reception is possible for example when a MAC layer makes multiple attempts for a transmission.

Therefore, when this timer expires, we are confident that the missed frame is no longer in transit and that the sender gave up.

$T_{BeforeNACK} = (N_{MACRetries} - 1) \text{ x } (T_{MAC} + T_{TransData} + T_{Prop})$

2.15.2 T_{NACK}

After sending a NACK to the MAC layer, a node expects to start receiving soon the requested missed frames. If it did not receive any before T_{NACK}, it sends the NACK again. Therefore, when this timer expires, we are confident that either the NACK was lost, or that one of the missed frames failed in its retransmission.

$T_{NACK} = N_{MACRetries} \text{ x } (T_{MAC} + T_{TransNACK} + T_{Prop}) + N_{MACRetries} \text{ x } (T_{MAC} + T_{TransData} + T_{Prop})$

$= N_{MACRetries} \text{ x } (2T_{MAC} + 2T_{Prop} + T_{TransNACK} + T_{TransData})$

2.15.3 N_{FrACK}

This is a configurable parameter. It is the maximum number of frames between each request for a positive acknowledgement. The higher its value, the lower the overhead is since less positive acknowledgements are sent. However, the memory consumption of the transport protocol is proportional to this value. Also, lower N_{FRACK} will lower the worse case frame latency.

The value of N_{FRACK} will be determined empirically or estimated based on memory availability and worse traffic and topology predictions.

2.15.4 T_{FrACK}

Under different conditions, a node requests a positive acknowledgment. One of them is after sending a certain number of frames. If there are not enough subsequent frames, a positive acknowledgement is required after a while to ensure proper transmission of previous frames. This is the purpose of the T_{FRACK} timer.

$T_{FRACK} \approx N_{FrACK} \text{ x } (T_{MAC} + T_{TransData} + T_{Prop}) \text{ x } N_{MACRetries} \text{ x } ß$

Where ß is a configurable or dynamic parameter.

If ß is configurable, the following guidelines should be used:

- $0 < ß \leq 1$
- ß should be set to values close to one on networks where high errors rates are expected to be typical.
- ß should be set to around 0.5 on networks where 50% frame errors are expected to be typical.
- ß should be set to a value lower than 0.5 on networks where low error rate is expected to be typical.

If ß is dynamic, then the MAC layer should inform the transport protocol about current average number or retransmission per frame. Then, ß is set to

- ß = (Current average Number of MAC retransmission per frame) / $N_{MACRetries}$

2.15.5 T_{ACK}

When this timer expires, a node that requested a positive ACK retransmits the frame in which the ACKREQ bit was set. In other words, after requesting an ACK, a node should have received it within T_{ACK}, otherwise it requests it again.

T_{ACK} $= N_{MACRetries}$ x $(T_{MAC} + T_{TransData} + T_{Prop}) + N_{MACRetries}$ x $(T_{MAC} + T_{TransACK} + T_{Prop})$
$= N_{MACRetries}$ x $(2T_{MAC} + 2T_{Prop} + T_{TransACK} + T_{TransData})$

2.15.6 $T_{ACKTopologyChange}$

After requesting an ACK just after a topology change to the next hop to the destination, a node should have received the ACK within $T_{ACKTopologyChange}$, otherwise it requests it again. This replaces momentarily T_{ACK}.

$T_{ACKTopologyChange} = N_{ACK}$ x $T_{ACK} + T_{ACK} = T_{ACK}$ x $(1 + N_{ACK})$

2.15.7 $T_{Reassembly}$

In unreliable mode, a packet not fully received is re-assembled if the $T_{Reassembly}$ timer expires, whether or not there are missed frames in the packet. It should be long enough such that we are sure the frames are no longer in transit.

The following guideline is given:
$T_{Reassembly} > T_{MAC} + T_{TransData} + T_{Prop}$

2.15.8 N_{NACK}

This is the maximum number of attempts of retransmissions of a frame by receiver, through NACKs. The value of this number should be high enough to ensure high reliability. It should be tested that this timer almost never expires. This counter should be high enough to handle frame errors, but it does not need to handle topology changes since this is handled by another mechanism.

2.15.9 N_{ACK}

This is the maximum number of times that a request for a positive acknowledgment can be attempted for a given frame. It should be tested that this almost never expires.

2.15.10 $T_{TopologyChange}$

In the case of an implementation where forwarders are not informed of topology changes, they can only count on themselves to conclude that they no longer are involved in a transmission. This decision is taken if no activity is detected for a certain period on a given unique pair of source/destination. This is analysed separately for reliable and unreliable traffic – but expiry of $T_{TopologyChange}$ for both types of traffic must happen to conclude to a topology change). When $T_{TopologyChange}$ expires, a node is confident that it is no longer involved in any transmission between a pair of source/destination.

The following guideline is given:
$T_{TopologyChange} >>> T_{MAC} + T_{TransData} + T_{Prop}$
$T_{TopologyChange} > T_{FRACK} + (T_{MAC} + T_{TransData} + T_{Prop})$ x $N_{MACRetries}$ x N_{ACK}

> (We know that after T_{FRACK} an ACK is requested in the next frame. We add to it the maximum time it can take for that frame to reach the forwarder – if there are no other frames to come, the last one would have been sent with the ACKREQ bit anyway)

3 Acronyms

ACK	Positive Acknowledgement
COTS	Commercial Off-The-Shelf Equipment
CRCTP	CRC Transport Protocol
L2N	Low bandwidth, Lossy Network
MAC	Medium Access Layer
NACK	Negative Acknowledgement
OSI	Open Systems Interconnection
PHY	Physical Layer
PSFQ	Pump Slowly, Fetch Quickly
SASNet	Self-Healing Autonomous Sensor Network
TCP	Transmission Control Protocol
WRSN	Wireless Radiation Sensor Network

4 References

[1] M. Déziel, F. Daigle, "Reliable Transport Protocol for SASNet Level-1 Network: Requirements, Literature Survey & Simulation/Evaluation of a Candidate Protocol", CRC Technical Memo # VPNT 2010/01, SASNet project, Communications Research Centre (CRC), Canada, 2010.

[2] J. Postel, "User Datagram Protocol", Internet Standard RFC768, Internet Engineering Task Force (IETF), August 1980.

[3] "Transmission Control Protocol - DARPA Internet Program - Protocol Specification", Internet Standard RFC793, Internet Engineering Task Force (IETF), September 1981.